Worksheet 5: Splines and Fourier Series

May 11th, 2016

General Remarks

- The deadline for handing in the worksheets is **Tuesday, May 24th, 2016, 10:00**.
- On this worksheet, you can achieve a maximum of 10 points.
- To hand in your solutions, send an email to `mkuron@icp.uni-stuttgart.de`.
- Please try to only hand in a single file that contains your program code for all tasks. If you are asked to answer questions, you should do so in a comment in your code file or a text block in your IPython notebook. If you need to include an equation or graph, you can do that in your IPython notebook, or you may hand in a separate PDF document with all your answers, graphs and equations.

Task 5.1: Spline Interpolation (6 points)

In this task, you should spline-interpolate the following functions are used on the specified domains:

Name	Definition	Domain
Sine Function	$f(x) = \sin x$	$[0, 2\pi]$
Runge Function	$g(x) = \frac{1}{1+x^2}$	$[-5, 5]$
Lennard-Jones Function	$h(x) = x^{-12} - x^{-6}$	$[1, 5]$

- **5.1.1** (2 points) Write a class `CubicSplineInterpolation` that implements cubic spline interpolation. As in the previous tasks, the class shall provide `__init__(self,args)` to initialize the interpolation and `__call__(self,x)` to compute the value of the interpolating function at `x`. On the boundaries, set the second derivative of the spline to 0. Note that the splines in this class will not be identical to the splines in the previous task, as SciPy uses different boundary conditions. Use the class to do the same plots as in the demo.

  Hint The method `SplineInterpolation.__init__()` has to compute the spline coefficients by first generating the defining linear equations (from the lecture script) and then solving them using `scipy.linalg.solve`.

- **5.1.2** (2 points) Derive the equations for the *quadratic spline*, where the spline polynomial is defined as $P_i(x) = y_i + m_i(x - x_i) + M_i(x - x_i)^2$. The conditions for the quadratic spline are that it has the function value at the supporting points x_i and x_{i+1} and that the first derivative of the spline at x_{i+1} is the same for P_i and P_{i+1}.

- **5.1.3** (2 points) Write a class `QuadraticSplineInterpolation` that implements the quadratic spline interpolation. Redo the same plots as in the previous tasks with quadratic splines.

Task 5.2: Fourier Transform in NumPy (4 points)

In this task, we will decompose the periodic Runge function $f(x) = \frac{1}{1+\cos^2(x)}$ on the domain $[0, 2\pi]$ and the Lennard-Jones-Function $g(x) = x^{-12} - x^{-6}$ on the domain $[1, 5]$ into a Fourier series.

On the next worksheet, you will implement the discrete fourier transform (DFT) and fast fourier transform (FFT) algorithms yourself, but for this week it is sufficient to use the existing implementations in NumPy.

- **5.2.1** (3 points) Write a Python program that does the following for the functions $f(x)$ and $g(x)$:

  1. It creates a data series of 2048 equidistant points on the respective domains and their function values.
  2. It uses the Python function `numpy.fft.fft()`, to compute the Fourier coefficients of the data series.
  3. It sets the Fourier coefficients of $|k| > k_{\max}$ where $k_{\max} \in \{5, 10, 50\}$ to 0, *i.e.* it truncates the Fourier series at $k_{\max}$.
  4. It back transforms the truncated Fourier series to real space with help of the Python function `numpy.fft.ifft()`. The output vector is the *reconstructed function.*
  5. It creates a plot of the function and the different reconstructed functions.

  Hints

  – Have a look at how the Fourier coefficients are stored to understand what coefficients you have to zero out. Read the docs of `numy.fft`.

  – Note that in principle, it is not necessary to store the Fourier coefficients that are 0. Therefore, truncating the Fourier series can be seen as a very effective data compression: instead of storing 2048 function values, it is enough to store just 50 Fourier coefficients to be able to reconstruct the data series to a high degree of accuracy.

- **5.2.2** (1 point) Why does the reconstruction of the function $g(x)$ show artifacts at the right boundary even when 50 Fourier coefficients are used, while this is not the case for function $f(x)$ even when only 10 Fourier coefficients are used?